Типы в языках программирования
Лекция 8. Экзистенциальные типы

Денис Николаевич Москвин

ИТМО, корпоративная магистратура JetBrains
Разработка ПО / Software Engineering

14.04.2021

План лекции

1. Определение экзистенциальных типов
2. Абстракция через экзистенцию
3. Кодирование экзистенциальных типов в System F
4. Кодирование экзистенциальных типов в Haskell

План лекции

Мотивация

- *Универсальные типы с логической точки зрения:*
  терм типа $\forall\alpha.\,\sigma$ является значением, которому можно приписать тип $[\alpha \mapsto \tau]\,\sigma$ при любом выборе τ.
- *Универсальные типы операционно:*
  термом типа $\forall\alpha.\,\sigma$ является функция $\lambda\alpha^*.\,M$, которая переводит произвольный тип τ в терм типа $[\alpha \mapsto \tau]\,\sigma$.

Мотивация

- *Универсальные типы с логической точки зрения:*
  терм типа $\forall\alpha.\,\sigma$ является значением, которому можно приписать тип $[\alpha \mapsto \tau]\,\sigma$ при любом выборе τ.
- *Универсальные типы операционно:*
  термом типа $\forall\alpha.\,\sigma$ является функция $\lambda\alpha^*.\,M$, которая переводит произвольный тип τ в терм типа $[\alpha \mapsto \tau]\,\sigma$.
- *Экзистенциальные типы логически:*
  терм типа $\exists\alpha.\,\sigma$ является значением, которому можно приписать тип $[\alpha \mapsto \tau]\,\sigma$ для некоторого выбора τ.
- *Экзистенциальные типы операционно:*
  термом типа $\exists\alpha.\,\sigma$ является пара $\{\tau, M\}$, состоящая из некоторого типа τ и терма M типа $[\alpha \mapsto \tau]\,\sigma$.

Экзистенциальный тип

- Значение типа $\exists\alpha.\,\sigma$ рассматривается как *пакет* (или *модуль*) $\{\tau, M\}$ с одним (скрытым) компонентом-типом и одним компонентом-термом.
- Тип τ называют *скрытым типом представления* (hidden representation type), или *типом-свидетелем* (witness type) пакета.
- Например, пакет $\{\mathbb{N}, \{a = 5, f = \lambda x^{\mathbb{N}}.\,\mathtt{succ}\ x\}\}$ имеет экзистенциальный тип $\exists\alpha.\{a^{\alpha}, f^{\alpha\to\alpha}\}$.

Экзистенциальный тип

- Значение типа $\exists\alpha.\,\sigma$ рассматривается как *пакет* (или *модуль*) $\{\tau, M\}$ с одним (скрытым) компонентом-типом и одним компонентом-термом.
- Тип τ называют *скрытым типом представления* (hidden representation type), или *типом-свидетелем* (witness type) пакета.
- Например, пакет $\{\mathbb{N}, \{a = 5, f = \lambda x^{\mathbb{N}}.\,\mathtt{succ}\ x\}\}$ имеет экзистенциальный тип $\exists\alpha.\{a^{\alpha}, f^{\alpha\to\alpha}\}$.
- Однако, с тем же успехом можно этому пакету приписать экзистенциальный тип $\exists\alpha.\{a^{\alpha}, f^{\alpha\to\mathbb{N}}\}$.
- Автоматическое введение экзистенциального типа невозможно, это дело рук программиста.
- Альтернативный синтаксис упаковки $\mathtt{pack}\ \alpha = \tau\ \mathtt{with}\ M$.

Введение экзистенциального типа

- Экзистенциальные типы вводим приписыванием

$$\{\mathbb{N}, \{a = 5, f = \lambda x^{\mathbb{N}}.\, \mathrm{succ}\ x\}\}\ \mathtt{as}\ \exists\alpha.\, \{a^{\alpha}, f^{\alpha\to\alpha}\}$$

- Правило типизации для введения квантора существования

Типизация

$$\frac{\Gamma \vdash M : [\alpha \mapsto \tau]\,\sigma}{\Gamma \vdash \{\tau, M\}\ \mathtt{as}\ \exists\alpha.\,\sigma\ :\ \exists\alpha.\,\sigma}\quad (\mathrm{T-Pack})$$

- Разные типы-свидетели могут населять один тип

$$\{\mathbb{N}, 0\}\ \mathtt{as}\ \exists\alpha.\,\alpha\quad :\ \exists\alpha.\,\alpha$$

$$\{\mathbb{B}, \mathtt{true}\}\ \mathtt{as}\ \exists\alpha.\,\alpha\quad :\ \exists\alpha.\,\alpha$$

Элиминация экзистенциального типа

- Элиминация пакета подобна директиве `open` или `import`.
- Можно использовать содержимое модуля в каком-то другом месте программы, при этом абстрактная природа типа модуля сохраняется.

Типизация

$$\frac{\Gamma \vdash M : \exists \alpha.\, \sigma \qquad \Gamma, \alpha^*, x^\sigma \vdash N : \rho}{\Gamma \vdash \texttt{let}\ \{\alpha, x\} = M\ \texttt{in}\ N : \rho} \quad (T - \mathrm{Unpack})$$

- Здесь α и x связывают типовую и термовую компоненты пакета и используются при вычислении N.
- Иногда используют синтаксис $\texttt{open}\ M\ \texttt{as}\ \{\alpha, x\}\ \texttt{in}\ N$.

Элиминация экзистенциального типа (примеры)

Типизация

$$\frac{\Gamma \vdash M : \exists\alpha.\,\sigma \qquad \Gamma, \alpha^*, x^\sigma \vdash N : \rho}{\Gamma \vdash \texttt{let}\ \{\alpha, x\} = M\ \texttt{in}\ N : \rho}\ (T - \text{Unpack})$$

$$p1 \equiv \{\mathbb{N}, \{a = 5, f = \lambda x^{\mathbb{N}}.\,\texttt{succ}\ x\}\}\ \texttt{as}\ \exists\alpha.\{a^\alpha, f^{\alpha\to\mathbb{N}}\}$$

$$e1 \equiv \texttt{let}\ \{\alpha, x\} = p1\ \texttt{in}\ (x.f)\ (x.a)$$
$$e1 : \mathbb{N}$$
$$e1 \twoheadrightarrow_\beta 6$$

$$e2 \equiv \texttt{let}\ \{\alpha, x\} = p1\ \texttt{in}\ (\lambda y^\alpha.\,(x.f)\ y)\ (x.a)$$
$$e2 : \mathbb{N}$$
$$e2 \twoheadrightarrow_\beta 6$$

Последний пример показывает, что в теле формы устранения можно также упоминать типовую переменную α.

Элиминация экзистенциального типа (запреты)

Типизация

$$\frac{\Gamma \vdash M : \exists\alpha.\,\sigma \qquad \Gamma, \alpha^*, x^\sigma \vdash N : \rho}{\Gamma \vdash \texttt{let}\ \{\alpha, x\} = M\ \texttt{in}\ N : \rho}\ (T - \mathrm{Unpack})$$

Тип представления пакета при проверке в теле распаковки должен оставаться абстрактным:

$$p1 \equiv \{\mathbb{N}, \{a = 5, f = \lambda x^{\mathbb{N}}.\,\mathtt{succ}\ x\}\}\ \mathtt{as}\ \exists\alpha.\{a^\alpha, f^{\alpha\to\mathbb{N}}\}$$

$$e3 \equiv \mathtt{let}\ \{\alpha, x\} = p1\ \mathtt{in}\ \mathtt{succ}\ (x.a)$$
$$e3 : \mathbf{Error}$$

$$p2 \equiv \{\mathbb{B}, \{a = \mathtt{true}, f = \lambda x^{\mathbb{B}}.\,42\}\}\ \mathtt{as}\ \exists\alpha.\{a^\alpha, f^{\alpha\to\mathbb{N}}\}$$

Ошибка возвращается справедливо, p2 имеет тот же тип!

Элиминация экзистенциального типа (запреты 2)

$$\frac{\Gamma \vdash M : \exists\alpha.\,\sigma \qquad \Gamma, \alpha^*, x^\sigma \vdash N : \rho}{\Gamma \vdash \mathtt{let}\ \{\alpha, x\} = M\ \mathtt{in}\ N : \rho}\ (\mathrm{T-Unpack})$$

В контексте заключения правила типовой переменной α нет, следующий пример даст ошибку области видимости:

$$\mathtt{p1} \equiv \{\mathbb{N}, \{a = 5, f = \lambda x^{\mathbb{N}}.\,\mathtt{succ}\ x\}\}\ \mathtt{as}\ \exists\alpha.\{a^\alpha, f^{\alpha\to\mathbb{N}}\}$$

$$\mathtt{e4} \equiv \mathtt{let}\ \{\alpha, x\} = \mathtt{p1}\ \mathtt{in}\ (x.a)$$
$$\mathtt{e4} : \mathbf{Error}$$

Это можно форсировать, уточнив правило типизации

$$\frac{\Gamma \vdash M : \exists\alpha.\,\sigma \qquad \Gamma, \alpha^*, x^\sigma \vdash N : \rho \qquad \Gamma \Vdash \rho : *}{\Gamma \vdash \mathtt{let}\ \{\alpha, x\} = M\ \mathtt{in}\ N : \rho}\ (\mathrm{T-Unpack})$$

Расширение System F

Новые синтаксические формы

$$t ::= x \mid t\,t \mid t\,\tau \mid \lambda x^{\tau}.\,t \mid \Lambda\alpha.\,t \mid \{\tau, t\}\ \mathtt{as}\ \tau \mid \mathtt{let}\ \{\alpha, x\} = t\ \mathtt{in}\ t$$

$$v ::= \ldots \mid \{\tau, v\}\ \mathtt{as}\ \tau$$

$$\tau ::= \alpha \mid \tau \to \tau \mid \forall\alpha.\,\tau \mid \exists\alpha.\,\tau$$

$$\Gamma ::= \varnothing \mid \Gamma, x{:}\tau \mid \Gamma, \alpha : *$$

Синтаксис термов расширяется упаковкой и распаковкой, вводится значение-пакет и экзистенциальный тип.

Образование типов

$$\frac{\alpha{:}* \in \Gamma}{\Gamma \Vdash \alpha{:}*} \quad (\mathrm{TG-Ini})$$

$$\frac{\Gamma \Vdash \sigma : * \qquad \Gamma \Vdash \tau : *}{\Gamma \Vdash \sigma \to \tau : *} \quad (\mathrm{TG-Arrow})$$

$$\frac{\Gamma, \alpha{:}* \Vdash \tau : *}{\Gamma \Vdash \forall \alpha.\, \tau : *} \quad (\mathrm{TG-ForAll})$$

$$\frac{\Gamma, \alpha{:}* \Vdash \tau : *}{\Gamma \Vdash \exists \alpha.\, \tau : *} \quad (\mathrm{TG-Exists})$$

Правила типизации

Типизация (новые правила)

$$\frac{\Gamma \vdash M : [\alpha \mapsto \tau]\,\sigma}{\Gamma \vdash \{\tau, M\}\ \mathtt{as}\ \exists\alpha.\,\sigma\ :\ \exists\alpha.\,\sigma} \qquad (\mathrm{T-Pack})$$

$$\frac{\Gamma \vdash M : \exists\alpha.\,\sigma \qquad \Gamma, \alpha^{*}, x^{\sigma} \vdash N : \rho \qquad \Gamma \Vdash \rho : *}{\Gamma \vdash \mathtt{let}\ \{\alpha, x\} = M\ \mathtt{in}\ N : \rho} \qquad (\mathrm{T-Unpack})$$

Правила вычисления

Вычисление (новые правила)

$$\frac{t \longrightarrow t'}{\{\tau, t\}\ \mathtt{as}\ \sigma \longrightarrow \{\tau, t'\}\ \mathtt{as}\ \sigma} \quad (E - Pack)$$

$$\mathtt{let}\ \{\alpha, x\} = (\{\tau, v\}\ \mathtt{as}\ \sigma)\ \mathtt{in}\ t \longrightarrow [\alpha \mapsto \tau]\,[x \mapsto v]\,t \quad (E - UnpackPack)$$

$$\frac{t_1 \longrightarrow t_1'}{\mathtt{let}\ \{\alpha, x\} = t_1\ \mathtt{in}\ t_2 \longrightarrow \mathtt{let}\ \{\alpha, x\} = t_1'\ \mathtt{in}\ t_2} \quad (E - Unpack)$$

Второе правило аналогично компоновке модулей: символические имена (α и x), заменяются реальным содержимым подключаемого модуля.

План лекции

Абстрактные типы данных

- Абстрактный тип данных (ADT) состоит из
  1. имени типа A;
  2. типа конкретного представления T;
  3. реализации некоторых операций над типом T;
  4. барьера абстракции.

```
ADT counter =
  type Counter
  representation Nat
  signature
    new : Counter,
    get : Counter → Nat,
    inc : Counter → Counter;
  operations
    new = 1,
    get = λi:Nat. i,
    inc = λi:Nat. succ(i);
counter.get (counter.inc counter.new);
```

АТД на экзистенциальных типах

```
counterADT =
  {Nat,
    {new = 1,
    get = λi:Nat. i,
    inc = λi:Nat. succ(i)}}
 as ∃Counter.
      {new: Counter,
      get: Counter → Nat,
      inc: Counter → Counter}
```

```
  let {Counter, counter} = counterADT in
    counter.get (counter.inc counter.new);
▷ 2 : Nat
```

Замена внутреннего представления на, скажем, запись с полем типа Nat не повлияет на использование.

План лекции

Кодирование пар в System F

Для конкретных типов τ и σ пара значений кодируется так

$$\begin{array}{lcl} \mathtt{Pair}\tau\sigma & \equiv & \forall\beta.\,(\tau \to \sigma \to \beta) \to \beta \\ \\ \mathtt{pair} & \equiv & \lambda x^{\tau} y^{\sigma}.\,\Lambda\alpha.\,\lambda f^{\tau\to\sigma\to\alpha}.\,f\,x\,y \\ \\ \mathtt{fst} & \equiv & \lambda p^{\mathtt{Pair}\tau\sigma}.\,p\,\tau\,(\mathbf{K}\,\tau\,\sigma) \\ \mathtt{snd} & \equiv & \lambda p^{\mathtt{Pair}\tau\sigma}.\,p\,\sigma\,(\mathbf{K}_*\,\tau\,\sigma) \end{array}$$

Идея: закодировать пару из типа и значения аналогично типу пары значений

$$\begin{array}{lcl} \mathtt{Pair}\tau\sigma & \equiv & \forall\beta.\,(\tau \to \sigma \to \beta) \to \beta \\ \\ \exists\alpha.\,\sigma & \equiv & \forall\beta.\,(\forall\alpha.\ \ \sigma \to \beta) \to \beta \end{array}$$

Кодирование упаковки пакета

Экзистенция через всеобщность

$$\exists\alpha.\,\sigma \;\equiv\; \forall\beta.\,(\forall\alpha.\,\sigma \to \beta) \to \beta$$

Типизация упаковки

$$\frac{\Gamma \vdash M : [\alpha \mapsto \tau]\,\sigma}{\Gamma \vdash \{\tau, M\}\ \mathbf{as}\ \exists\alpha.\,\sigma\ :\ \exists\alpha.\,\sigma} \quad (\mathrm{T-Pack})$$

Кодирование упаковки пакета

$$\{\tau, M\}\ \mathbf{as}\ \exists\alpha.\,\sigma \;\equiv\; \Lambda\beta.\,\lambda f^{\forall\alpha.\,\sigma\to\beta}.\,f\,\tau\,M$$

Здесь f, будучи примененным к τ в теле абстракции, приобретает ожидаемый для следующей аппликации тип $([\alpha \mapsto \tau]\,\sigma) \to \beta$.

Кодирование распаковки пакета

Экзистенция через всеобщность

$$\exists\alpha.\,\sigma \;\equiv\; \forall\beta.\,(\forall\alpha.\,\sigma \to \beta) \to \beta$$

Типизация распаковки

$$\frac{\Gamma \vdash M : \exists\alpha.\,\sigma \qquad \Gamma, \alpha^*, x^\sigma \vdash N : \rho}{\Gamma \vdash \mathtt{let}\ \{\alpha, x\} = M\ \mathtt{in}\ N : \rho}\quad (\mathrm{T-Unpack})$$

Кодирование распаковки пакета

$$\mathtt{let}\ \{\alpha, x\} = M\ \mathtt{in}\ N \;\equiv\; M\,\rho\,(\Lambda\alpha.\,\lambda x^\sigma.\,N)$$

В скобках делается универсальное замыкание N, которое используется для элиминации зависимой пары, аналогично $\mathbf{K}$ или $\mathbf{K}_*$ в случае обычной пары.

План лекции

Экзистенциальные типы в Haskell

```
{-# LANGUAGE ExistentialQuantification #-}
module Existentials where

data Obj = forall a. Show a ⇒ Obj a

xs :: [Obj]
xs = [Obj 42, Obj True, Obj 'c']
```

```
GHCi> foldr (λ (Obj x) s → show x ++ s) "" xs
"42True'c'"
```

Экзистенциальные типы в Haskell (2)

```
data Rec = forall a. Rec { a :: a, f :: a → Int }

p1,p2 :: Rec
p1 = Rec 5 succ
p2 = Rec True (λ _ → 42)

f1 :: Rec → Int
f1 (Rec val fun) = fun val
```

```
GHCi> f1 p1
6
GHCi> f1 p2
42
```

Экзистенциальные типы в Haskell (3)

Включив `RankNTypes`, можно задать кодирование и декодирование в универсальные функциональные типы

```
fromObj :: Obj → forall r.(forall a.Show a⇒a→r) → r
fromObj (Obj x) k = k x

toObj :: (forall r.(forall a.Show a⇒a→r) → r) → Obj
toObj f = f Obj
```

```
GHCi> :t fromObj (Obj 5)
fromObj (Obj 5) :: (forall a. Show a ⇒ a → r) → r
GHCi> fromObj (Obj 5) (λ a → show a ++ show a)
"55"
```

Экзистенциальные типы в Haskell (4)

```
fromRec :: Rec → forall r.(forall a.a→(a→Int)→r) → r
fromRec = undefined
toRec :: (forall r.(forall a.a→(a→Int)→r) → r) → Rec
toRec = undefined
```

```
GHCi> :t fromRec p1
fromRec p1 :: (forall a. a → (a → Int) → r) → r
GHCi> fromRec p1 (λ val fun → fun val)
6
GHCi> fromRec p2 (λ val fun → fun val)
42
GHCi> :t toRec $ fromRec p1
toRec $ fromRec p1 :: Rec
GHCi> f1 $ toRec $ fromRec p1
6
```